2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

Particle Computation: Device Fan-out and Binary Memory

Hamed Mohtasham Shad, Rose Morris-Wright, Erik D. Demaine, Sándor P. Fekete, Aaron T. Becker

***Abstract*— We present fundamental progress on the computational universality of swarms of micro- or nano-scale robots in complex environments, controlled not by individual navigation, but by a uniform global, external force. Consider a 2D grid world, in which all obstacles and robots are unit squares, and for each actuation, robots move maximally until they collide with an obstacle or another robot. In previous work, we demonstrated components of *particle computation* in this world, designing obstacle configurations that implement AND and OR logic gates: by using dual-rail logic, we designed NOT, NOR, NAND, XOR, XNOR logic gates. However, we were unable to design a FAN-OUT gate, which is necessary for simulating the full range of complex interactions that are present in arbitrary digital circuits. In this work we resolve this problem by proving unit-sized robots *cannot* generate a FAN-OUT gate. On the positive side, we resolve the missing component with the help of 2×1 robots, which can create fan-out gates that produce multiple copies of the inputs. Using these gates we are able to establish the full range of computational universality as presented by complex digital circuits. As an example we connect our logic elements to produce a 3-bit counter. We also demonstrate how to implement a data storage element.**

I. Introduction

One of the exciting new directions of robotics is the design and development of micro- and nanorobot systems, with the goal of letting a massive swarm of robots perform complex operations in a complicated environment. Due to scaling issues, individual control of the involved robots becomes physically impossible: while energy storage capacity drops with the third power of robot size, medium resistance decreases much slower. As a consequence, current micro- and nanorobot systems with many robots are steered and directed by an external force that acts as a common control signal [1]–[7]. These common control signals include global magnetic or electric fields, chemical gradients, and turning a light source on and off.

Clearly, having only one global signal that uniformly affects all robots at once poses a strong restriction on the ability of the swarm to perform complex operations. The only hope for breaking symmetry is to use interactions between the robot swarm and obstacles in the environment. The key challenge is to establish if interactions with obstacles are sufficient to perform complex operations, ideally by analyzing the complexity of possible logical operations. In previous work [8]–[10], we were able to demonstrate how

H. Mohtasham mohtasham@gmail.com, R. Morris-Wright rmorriswright@gmail.com, E. Demaine is with the Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA 02139, USA, edemaine@mit.edu, S. Fekete is with the Dept. of Computer Science, TU Braunschweig, Mühlenpfordtstr. 23, 38106 Braunschweig, Germany, s.fekete@tu-bs.de, A. Becker is with the Dept. of Electrical and Computer Engineering, University of Houston, Houston, TX 70004, USA atbecker@uh.edu.

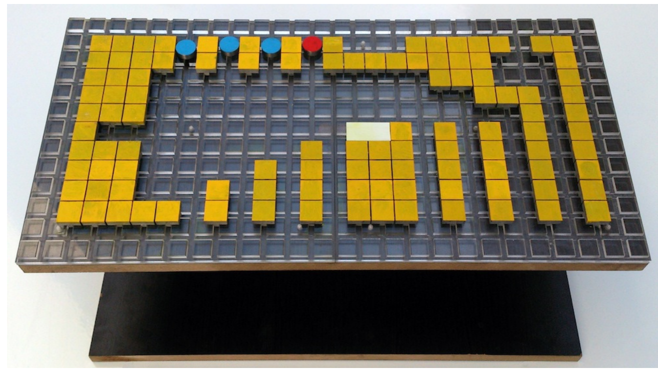

Fig. 1. Gravity-fed hardware implementation of particle computation. The reconfigurable prototype is setup as a FAN-OUT gate using a 2×1 robot (white). This paper proves that such a gate is impossible using only 1×1 robots. See the demonstrations in the video attachment http://youtu.be/EJSv8ny31r8.

a subset of logical functions can be implemented; however, devising a fan-out gate (and thus the ability to replicate and copy information) appeared to be prohibitively challenging. In this paper, we resolve this crucial question by showing that only using unit-sized robots is insufficient for achieving computational universality. Remarkably, adding a limited number of domino-shaped objects *is sufficient* to let a common control signal, mobile particles, and unit-sized obstacles simulate a computer. While this does not imply that large-scale computational tasks should be run on these particle computers instead of current electronic devices, it establishes that future nano-scale systems are able to perform arbitrarily complex operations *as part of the physical system*, instead of having to go through external computational devices.

A. Model

This paper builds on the techniques for controlling many simple robots with uniform control inputs presented in [8]–[10], using the following rules:

1) A planar grid *workspace* W is filled with a number of unit-square robots (each occupying one cell of the grid) and some fixed unit-square blocks. Each unit square in the workspace is either *free*, which a robot may occupy or *obstacle* which a robot may not occupy. Each square in the grid can be referenced by its Cartesian coordinates $\boldsymbol{x} = (x, y)$.
2) All robots are commanded in unison: the valid commands are "Go Up" (u), "Go Right" (r), "Go Down" (d), or "Go Left" (l).
3) The robots all move in the commanded direction until they hit an obstacle or a stationary robot. A *command sequence* $\boldsymbol{m}$ consists of an ordered sequence of moves m_k, where each $m_k \in \{u, d, r, l\}$ A representative

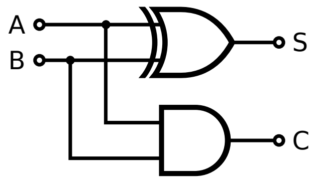

Fig. 2. The half adder shown above requires two copies of $\mathbf{A}$ and $\mathbf{B}$.

command sequence is $\langle u, r, d, l, l, d, r, u, \ldots\rangle$. We assume the area of W is finite and issue each command long enough for the robots to reach their maximum extent.

B. Dual-Rail Logic and FAN-OUT Gates

As shown in [8], AND and OR can be implemented with unit-size particles. However, particle logic is *conservative*—particles are neither created nor destroyed—and we were unable to implement a NOT gate. To implement NOT gates and other logic we used *dual-rail logic*, where two lines for each input are supplied to explicitly represent the variable and its complement [10]. Here we show that dual-rail logic is necessary, as single-rail logic is insufficient to produce a NOT gate.

The *fan out* of a logic gate output is the number of gate inputs it can feed or connect to. With particle logic, as demonstrated in [9], each logic gate output could fan out to only one gate. This is sufficient for *sum of products* and *product of sums* operations in CPLDs (complex programmable logic devices), but insufficient for more flexible architectures. Consider the half-adder shown in Fig. 2. The inputs $\mathbf{A}$ and $\mathbf{B}$ are needed to compute both the SUM and the CARRY bits, so the fanout of $\mathbf{A}$ and $\mathbf{B}$ is two. In this paper we prove the insufficiency of unit-sized particles for the implementation of fan-out gates, and design a fan-out gate using 2×1 particles.

C. Contributions

After a brief overview of related work, the contributions of this paper are as follows:

1) We prove the necessity of dual-rail logic for Boolean logic (Section III).
2) We prove the insufficiency of unit-size particles for gate fan-out (Section III).
3) We design FAN-OUT gates (Section IV-B).
4) We design memory latches (Section IV-C).
5) We present architecture for device integration, design a common clock sequence, and present a binary counter (Section IV-D).
6) We present the design and implementation of a large-scale particle computation prototype (Section V).

II. Related Work

Our efforts have similarities with *mechanical computers*, computers constructed from mechanical, not electrical components. For a fascinating nontechnical review, see [11]. These devices have a rich history, from the *Pascaline*, an adding machine invented in 1642 by a nineteen-year old Blaise Pascal; Herman Hollerith's punch-card tabulator in 1890; to the mechanical devices of IBM culminating in the 1940s. These devices used precision gears, pulleys, or electric motors to carry out calculations. Though our GRIDWORLD implementations are rather basic, we require none of these precision elements—merely unit-size obstacles, and sliding particles sized 2×1 and 1×1 for achieving computational universality.

Collision-Based Computing has been defined as *"computation in a structureless medium populated with mobile objects"*. For a survey of this area, see the excellent collection [12]. Early examples include the billiard-ball computer proposed by Fredkin and Toffoli using only spherical balls and a frictionless environment composed of elastic collisions with other balls and with angled walls [13]. Another popular example is Conway's *Game of Life*, a cellular automaton governed by four simple rules [14]. Cells live or die based on the number of neighbors. These rules have been examined in depth and used to design a Turing-complete computer [15]. Game of life scenarios and billiard-ball computers are fascinating, but lack a physical implementation. In this paper we present a collision-based system for computation and provide a physical implementation.

Sliding-Block Puzzles use rectangular tiles that are constrained to move in a 2D workspace. The objective is to move one or more tiles to desired locations. They have a long history. Hearn [16] and Demaine [17] showed tiles can be arranged to create logic gates, and used this technique to prove PSPACE complexity for a variety of sliding-block puzzles. Hearn expressed the idea of building computers from the sliding blocks—many of the logic gates could be connected together, and the user could propagate a signal from one gate to the next by sliding intermediate tiles. This requires the user to know precisely which sequence of gates to enable/disable. In contrast to such a hands-on approach, with our architecture we can build circuits, store parameters in memory, and then actuate the entire system in parallel using a global control signal.

Related Work on Programmable Matter: There is a wide range of interesting approaches to programmable matter. One model is the *abstract Tile-Assembly Model* (aTAM) by Winfree [18]–[20], which has sparked a range of theoretical and practical research. In this model, unit-sized pixels ("tiles") interact and bond with the help of differently labeled edges, eventually composing complex assemblies. Though the operations and final objectives in this model are different from our particle computation with global inputs (e.g., key features of the aTAM are that tiles can have a wide range of different edge types, and that they keep sticking together after bonding), there is a remarkable geometric parallelism to a key result of our present paper: While it is believed that at the most basic level of interaction (called *temperature 1*), computational universality *cannot be achieved* [21]–[23] in the aTAM with only unit-sized pixels, recent work [24] shows that computational universality *can be achieved* as soon as even slightly bigger tiles are used. This resembles the results of our paper, which shows that unit-size particles are insufficient for universal computation, while employing bigger particles suffices.

III. THEORY

First we provide terminology to define how robots interact with each other.

a) Definition of hit: During move m_k, robot a *hits* robot b if a is prevented from reaching location $\boldsymbol{x} = (x, y)$ because robot b occupies this location. Robot b's location at the end of m_k will be $\boldsymbol{x}$ and robot a's location at the end of m_k can be calculated as follows:

$$\begin{cases} (x-1, y) & \text{if } m_k = r \\ (x+1, y) & \text{if } m_k = l \\ (x, y-1) & \text{if } m_k = u \\ (x, y+1) & \text{if } m_k = d \end{cases}$$

b) Definition of path: As robot r travels from $\boldsymbol{s}$ to $\boldsymbol{g}$, it passes through a sequence of locations $\{\boldsymbol{s}, \ldots, \boldsymbol{g}\}$. We call this sequence of locations r's *path*.

Lemma 1: Given a workspace W, a command sequence $\boldsymbol{m}$, and a robot r traveling along a path beginning at location $\boldsymbol{s}$, this path can only be changed by introducing a new robot that r hits, or by removing one of the robots that r hit before.

Proof: By definition, the path is the sequence of locations occupied by the robot. The path is entirely determined by four factors: the starting location of the robot, the command sequence, obstacles in the workspace, and encounters with other robots already occupying free locations. If no hits with other robots are added or subtracted, then robot r will start in the same place, receive the same command sequence and encounter the same obstacles, so robot r's path will remain unchanged. Note that r's path is still unchanged if it is hit by another robot. According to our definition of a hit, if robot q hits r, then r's path is unchanged, while if robot r hits q then r's path is changed. ■

Now we show that adding more unit-sized robots cannot prevent a location from being occupied at the end of the command sequence.

Theorem 2: If given a workspace W and a command sequence $\boldsymbol{m}$ that moves a robot from a start location $\boldsymbol{s}$ to a goal location $\boldsymbol{g}$, adding additional robots anywhere in W at any stage of the command sequence cannot prevent $\boldsymbol{g}$ from being occupied at the conclusion of sequence $\boldsymbol{m}$.

Proof: Consider the effect of adding robot b to workspace W. If a never hits b, then by Lemma 1, a's path remains the same. Therefore, at the conclusion of $\boldsymbol{m}$, a occupies $\boldsymbol{g}$.

Now suppose a hits b. By the definition of a hit, b prevents a from reaching some location $\boldsymbol{x}$ because b already occupies this location. After the hit, the command sequence will continue and so robot b will continue on a's original path, following the same instructions and therefore ending up in the same location, $\boldsymbol{g}$, unless b hits yet another robot. By induction, we can clearly see that such additional robots will have the same effect. If b hits any other robot, this robot will continue on the original path. Thus by adding more robots, it is impossible to prevent some robot from occupying $\boldsymbol{g}$ at the conclusion of $\boldsymbol{m}$. ■

Inputs			*Outputs*			
A	$\overline{A}$	1	A	A	$\overline{A}$	$\overline{A}$
0	1	1	0	0	1	1
1	0	1	1	1	0	0

TABLE I

FAN-OUT OPERATION. THIS CANNOT BE IMPLEMENTED WITH 1×1 PARTICLES AND OBSTACLES. OUR TECHNIQUE USES 2×1 PARTICLES.

Corollary 3: A NOT gate without dual-rail inputs cannot be constructed.

Proof: By contradiction. A particle logic NOT gate without dual-rail inputs has one input at $\boldsymbol{s}$, one output at $\boldsymbol{g}$, an arbitrary, possibly zero, number of asserted inputs which are all initially occupied, and an arbitrary, possibly zero, number of waste outputs.

In order for the NOT gate conditions to be satisfied, given a command sequence $\boldsymbol{m}$:

1) if $\boldsymbol{s}$ is initially unoccupied, $\boldsymbol{g}$ must be occupied at the conclusion of $\boldsymbol{m}$
2) if $\boldsymbol{s}$ is initially occupied, $\boldsymbol{g}$ must be unoccupied at the conclusion of $\boldsymbol{m}$

By Theorem 2, if $\boldsymbol{s}$ initially unoccupied results in $\boldsymbol{g}$ being occupied by some robot r at the conclusion of $\boldsymbol{m}$, then the addition of a robot q at $\boldsymbol{s}$ cannot prevent $\boldsymbol{g}$ from being filled, resulting in a contradiction. ■

This shows that dual-rail logic is necessary for the formation of NOT gates.

Additionally, we show that 1×1 robots are insufficient to produce fan-out gates. To this end, we must examine the possibilities both when we add additional robots to the scenario, as well as when we remove robots.

Theorem 4: If given a workspace W and a command sequence $\boldsymbol{m}$ that moves two robots, r_1 and r_2, initially at $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, to respective goal locations $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$, then removing one robot results in either $\boldsymbol{g}_1$ or $\boldsymbol{g}_2$ being occupied at the conclusion of $\boldsymbol{m}$.

Proof: Without loss of generality, robot r_1 is removed. First suppose r_2 never hits r_1. Then the removal of r_1 will not affect the path of r_2. Robot r_2 has the same number of hits that it had before the removal of r_1 and so by Lemma 1, r_2 will follow the same path and occupy $\boldsymbol{g}_2$ at the conclusion.

Alternatively, suppose r_2 hit r_1 when r_1 was occupying location $\boldsymbol{x}$. Because r_1 is removed, it no longer occupies $\boldsymbol{x}$ during this move; because it was stopped in the common direction when being hit by r_2, robot r_2 gets stopped by this obstacle at location $\boldsymbol{x}$, previously occupied by r_1. r_2 now proceeds along the path previously traveled by r_1. Effectively, r_2 has replaced r_1 and follows r_1's path until it reaches $\boldsymbol{g}_1$. Successive hits between r_2 and r_1 in the original scenario are resolved in the same manner. ■

Corollary 5: A conservative dual-rail logic FAN-OUT gate cannot be constructed using only 1×1 robots.

Proof: We assume such a FAN-OUT gate exists and reach a contradiction. Consider a FAN-OUT gate W, dual-rail input locations $\boldsymbol{s}_a$, $\boldsymbol{s}_{\overline{a}}$, and dual-rail output locations $\{\boldsymbol{g}_{a_1}, \boldsymbol{g}_{a_2}, \boldsymbol{g}_{\overline{a}_1}, \boldsymbol{g}_{\overline{a}_2}\}$. Because particle logic is conservative there must also be one additional input location $\boldsymbol{s}_r$ and robot

r. A FAN-OUT gate implements the truth table shown in Table I. Given an arbitrary command sequence $\boldsymbol{m}$:

1) if $\boldsymbol{s}_a$ and $\boldsymbol{s}_r$ are initially occupied and $\boldsymbol{s}_{\overline{a}}$ vacant, at the conclusion of $\boldsymbol{m}$ then $\boldsymbol{g}_{a_1}$ and $\boldsymbol{g}_{a_2}$ are occupied and the locations $\boldsymbol{g}_{\overline{a}_1}$ and $\boldsymbol{g}_{\overline{a}_2}$ are vacant.
2) if $\boldsymbol{s}_a$ is initially vacant and $\boldsymbol{s}_{\overline{a}}$ and $\boldsymbol{s}_r$ are occupied, at the conclusion of $\boldsymbol{m}$ then g_{a_1} and g_{a_2} are vacant and the locations $g_{\overline{a}_1}$ and $g_{\overline{a}_2}$ are occupied.

We will now assume that condition 1, above, is the original scenario and add and subtract robots, applying theorems 2 and 4, to show that it is impossible to meet condition 2.

Assume condition 1. Robots a and r start at $\boldsymbol{s}_a$ and $\boldsymbol{s}_r$ respectively and at the conclusion of $\boldsymbol{m}$, the locations $\boldsymbol{g}_{a1}$ and $\boldsymbol{g}_{a2}$ are occupied. Now remove robot a. According to Thm. 4, either $\boldsymbol{g}_{a_1}$ or $\boldsymbol{g}_{a_2}$ must be occupied at the conclusion of $\boldsymbol{m}$. Suppose without loss of generality that $\boldsymbol{g}_{a_1}$ is filled. By Thm. 2, adding an additional robot at location $s_{\overline{a}}$ cannot prevent $\boldsymbol{g}_{a_1}$ from being filled. However, to meet condition 2, $\boldsymbol{g}_{a_1}$ must be vacant, thus no such gate is possible. ■

IV. Device and Gate Design

This section describes how the clock signal, logic gates, and wiring were designed.

A. Choosing a clock signal

The *clock sequence* is the ordered set of moves that are simultaneously applied to every particle in our workspace. As in digital computers, this sequence is universally applied and keeps all logic synchronized.

A clock sequence determines the basic functionality of each gate. To simplify implementation in the spirit of Reduced Instruction Set Computing (RISC), which uses a simplified set of instructions that run at the same rate, we want to use the same clock cycle for each gate and for *all* wiring. Our early work used a standard sequence $\langle d, l, d, r\rangle$. This sequence can be used to make AND, OR, XOR, and any of their inverses. This sequence can also be used for *wiring* to connect arbitrary inputs and outputs, as long as the outputs are below the inputs. Unfortunately, $\langle d, l, d, r\rangle$ cannot move any particles upwards. To connect outputs as inputs to higher-level logic requires an additional reset sequence that contains a $\langle u\rangle$ command. Therefore, including all four directions is a necessary condition for a valid clock sequence. The shortest sequence has four commands, each appearing once. We choose the sequence, $\langle d, l, u, r\rangle$, and by designing examples prove that this sequence is sufficient for logic gates, FAN-OUT gates, and wiring.

This clock sequence has the attractive property of being a clockwise (CW) rotation through the possible input sequences. One could imagine our particle logic circuit mounted on a wheel rotating about an axis parallel to the ground. If the particles were moved by the pull of gravity, each counter-clockwise revolution would advance the circuit by one clock cycle.

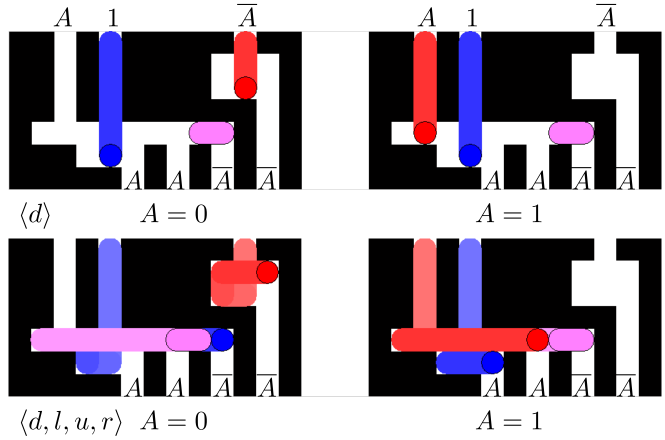

Fig. 3. A single input, two-output FAN-OUT gate. This gate requires a dual-rail input, a supply particle, and a 2×1 slider. The *clockwise* control sequence $\langle d, l, u, r\rangle$ duplicates the dual-rail input.

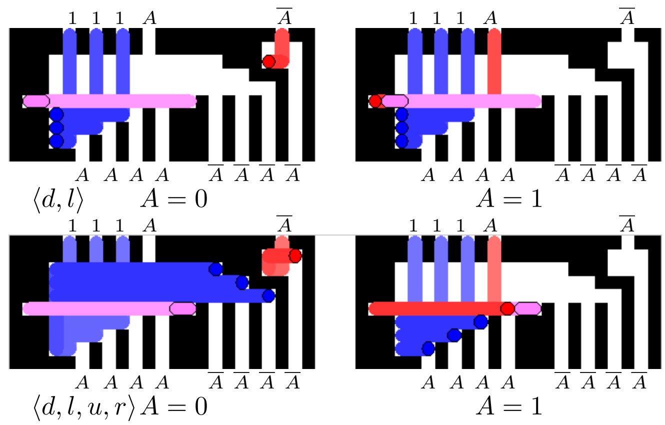

Fig. 4. The FAN-OUT gate can drive multiple outputs. Here a single input drives four outputs. This gate requires a dual-rail input, three supply particles, and a 2×1 slider. The *clockwise* control sequence $\langle d, l, u, r\rangle$ quadruples the dual-rail input.

B. A FAN-OUT Gate

A FAN-OUT gate with two outputs implements the truth table in Table I. This cannot be implemented with 1×1 particles and obstacles, by corollary 5. Our technique uses 2×1 particles. A single-input, two-output FAN-OUT gate is shown in Fig 3. This gate requires a dual-rail input, a supply particle, and a 2×1 slider. The *clockwise* control sequence $\langle d, l, u, r\rangle$ duplicates the dual-rail input.

The FAN-OUT gate can drive multiple outputs. In Fig. 4 a single input drives four outputs. This gate requires a dual-rail input, three supply particles, and a 2×1 slider. The *clockwise* control sequence $\langle d, l, u, r\rangle$ quadruples the dual-rail input. In general, an n-output FAN-OUT gate with control sequence $\langle d, l, u, r\rangle$ requires a dual-rail input, $n - 1$ supply particles, and one 2×1 slider. It requires an area of size $4(n + 1) \times 2(n + 1)$.

C. Data Storage

A general-purpose computer must be able to store data. A 2×1 particle enables us to construct a read/writable data storage for one bit. A single-bit data storage latch is shown in Fig. 6 and implements the truth table in Table II. By combining an n-out FAN-OUT gate shown in Fig 4 with n

Q	R	S	C	Q	Q_R	W_1	W_2	$\overline{Q}_R$
0	1	0	0	0	0	0	0	1
0	0	1	0	1	0	0	0	0
0	0	0	1	0	0	0	0	0
1	1	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1
1	0	0	1	0	0	1	1	0

TABLE II
A SINGLE-BIT DATA STORAGE LATCH WITH STATE Q.

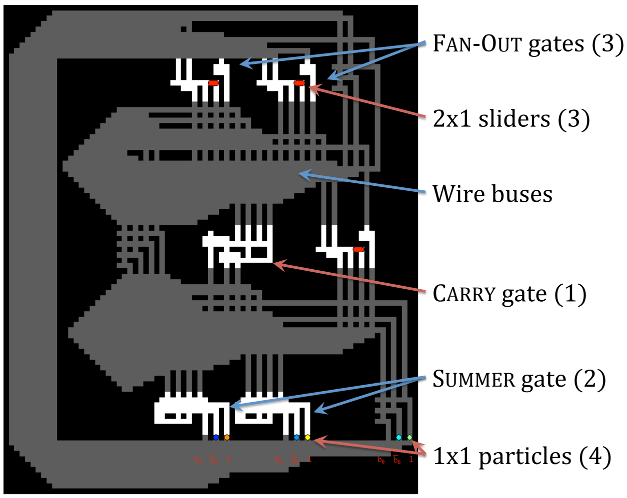

Fig. 5. A three-bit counter implemented with particles. The counter requires three FAN-OUT, two SUMMER, and one CARRY gates. Six 1×1 particles and three 2×1 particles are used. The counter has three levels of gates actuated by CW sequence $\langle d, l, u, r\rangle$ and requires three interconnection sequences $\langle d, l, u, r\rangle$, for a total of 24 moves. See video attachment.

data storage devices, we can implement an n-bit memory. To maintain *conservative* properties of the computer, i.e., the same number of robots enter and leave each gate, single-bit data storage latches must be used in pairs to record the state and its inverse.

D. A Binary Counter

Using the FAN-OUT gate from Section IV-B we can generate arbitrary Boolean logic. The half-adder from Fig. 2 requires a single FAN-OUT gate, one AND and one XOR gate.

We illustrate how several gates can be combined by constructing a binary counter, shown in Fig. 5.

E. Scaling Issues

Particle computation requires multiple clock cycles, workspace area for gates and interconnections, and many particles. In this section we analyze how these scale with the size of the counter.

a) gates: an n-bit counter requires $3(n-1)$ gates: n FAN-OUT gates, $n-1$ summers (XOR) gates, and $n-2$ carry (AND) gates.

b) particles: we require n 1×1 particles, one for each bit and n 2×1 particles, one for each FAN-OUT gate.

c) propagation delay: the counter requires n stages of logic, and n corresponding wiring stages. Each stage requires a complete clock cycle $\langle d, l, u, r\rangle$ for a total of $8n$ moves.

These requirements are comparable to a ripple-carry adder: the delay for n bits is n delays and requires $5(n-1)+2$ gates. Numerous other schemes exist to speed up the computation; however, using discrete gates allows us to use standard methods for translating a Boolean expression into gate-level logic. If speed was critical, instead of using discrete gates, we could engineer the workspace to directly compute the desired logic.

F. Optimal Wiring Schemes

With our current CW clock cycle, we cannot have outputs from the same column as inputs—outputs must be either one to the right, or three to the left. Choosing one of these results in horizontal shifts at each stage and thus requires spreading the logic gates horizontally. A better wiring scheme cycles through three layers that each shift right by one, followed by one layer that shifts left by three. We also want the wiring to be tight left-to-right. If our height is also limited, *wire buses*, shown in Fig. 5, provide a compact solution.

V. EXPERIMENT

A large-scale prototype was built as a physical implementation of particle computation with global inputs, shown in Fig. 1. The design is inspired by the gravity-fed logic maze *Tilt*^{TM} `http://www.thinkfun.com/tilt`. Figure 1 shows a functioning single-input, four-output FAN-OUT gate. The gate requires three 1×1 supply particles, shown in blue. The slider is white and the obstacles are yellow. The video attachment demonstrates how the gates are constructed and shows each variant in operation.

The experimental setup consists of Plexiglas square boards, which can be assembled together to form a bigger board. Each board has 12 vertical and 12 horizontal slots that constitute a checkered board. Each slot is 5mm in width and 4mm in depth. The distance between adjacent slots is 20mm. Different configurations can be achieved using 19mm$\times$19mm obstacles, which can be fitted in any slot junction. A clearance of 0.5mm on each side of the obstacle ensures that moving parts can slide past. There are two types of moving parts; 1×1 and 1×2 sliders. 1×1 sliders are 19.5mm diameter steel cylinders. Beneath this top cylinder is a cylindrical peg with a diameter of 4mm that guides the slider through the slot. 1×2 sliders are 20mm$\times$40mm rectangular steel parts with a rectangular guide at the bottom.

The board can rotate along two perpendicular axes and these rotations generate the four valid commands. Both types of sliders start moving at 12.5°. Experimental data on success rate as a function of tipping angle are shown in Fig. 7. We chose a rotation of 20° for each command to ensure that sliders slide reliably.

VI. CONCLUSION

In this paper we (1) proved the insufficiency of unit-size particles for gate fan-out; (2) established the necessity of dual-rail logic for Boolean logic; (3) designed FAN-OUT gates and memory latches by employing slightly different particles; (4) presented an architecture for device integration,

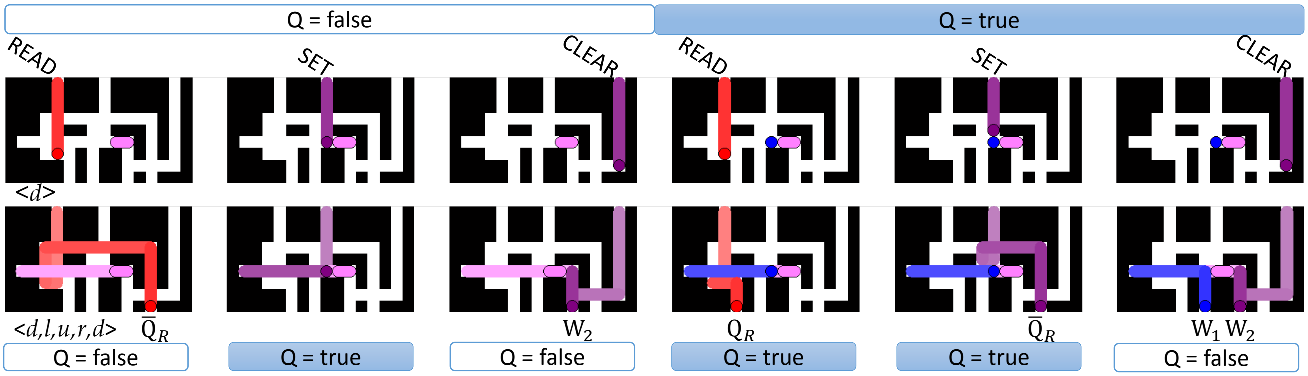

Fig. 6. A flip-flop memory. This device has three inputs, *Read*, *Set*, *Clear*, a state variable (shown in blue), and a 2×1 slider. Depending on which input is active, the *clockwise* control sequence $\langle d, l, u, r \rangle$ will read, set, or clear the memory.

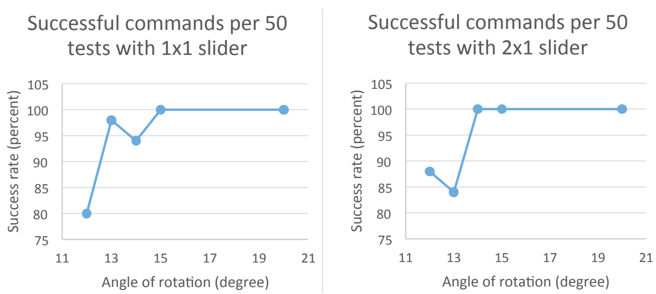

Fig. 7. Experimental data on slider reliability as a function of tip angle.

a common clock sequence, and a binary counter, and (5) implemented a large-scale prototype of particle computation.

This work, along with [8]–[10], introduces a new model for mechanical computation. Interesting applications will aim at nanoscale and microfluidics work.

References

[1] B. R. Donald, C. G. Levey, I. Paprotny, and D. Rus, "Planning and control for microassembly of structures composed of stress-engineered MEMS microrobots," *The International Journal of Robotics Research*, vol. 32, no. 2, pp. 218–246, 2013. [Online]. Available: http://ijr.sagepub.com/content/32/2/218.abstract

[2] P.-T. Chiang, J. Mielke, J. Godoy, J. M. Guerrero, L. B. Alemany, C. J. Villagómez, A. Saywell, L. Grill, and J. M. Tour, "Toward a light-driven motorized nanocar: Synthesis and initial imaging of single molecules," *ACS Nano*, vol. 6, no. 1, pp. 592–597, Feb. 2011.

[3] H.-W. Tung, D. R. Frutiger, S. Panè, and B. J. Nelson, "Polymer-based wireless resonant magnetic microrobots," in *IEEE International Conference on Robotics and Automation*, May 2012, pp. 715–720.

[4] E. Diller, J. Giltinan, and M. Sitti, "Independent control of multiple magnetic microrobots in three dimensions," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 614–631, 2013. [Online]. Available: http://ijr.sagepub.com/content/32/5/614.abstract

[5] W. Jing, N. Pagano, and D. Cappelleri, "A tumbling magnetic microrobot with flexible operating modes," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 5514–5519.

[6] Y. Ou, D. H. Kim, P. Kim, M. J. Kim, and A. A. Julius, "Motion control of magnetized tetrahymena pyriformis cells by magnetic field with model predictive control," *Int. J. Rob. Res.*, vol. 32, no. 1, pp. 129–139, Jan. 2013.

[7] D. de Lanauze, O. Felfoul, J.-P. Turcot, M. Mohammadi, and S. Martel, "Three-dimensional remote aggregation and steering of magnetotactic bacteria microrobots for drug delivery applications," *The International Journal of Robotics Research*, 11 2013. [Online]. Available: http://ijr.sagepub.com/content/early/2013/11/11/0278364913500543

[8] A. Becker, E. Demaine, S. Fekete, G. Habibi, and J. McLurkin, "Reconfiguring massive particle swarms with limited, global control," in *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS)*, Sep. 2013.

[9] A. Becker, E. Demaine, S. Fekete, and J. McLurkin, "Particle computation: Designing worlds to control robot swarms with only global signals," in *IEEE International Conference on Robotics and Automation (ICRA)*. Hong Kong: IEEE, May 2014, pp. 6751–6756.

[10] A. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin, "Reconfiguring massive particle swarms with limited, global control," in *Algorithms for Sensor Systems*, ser. Lecture Notes in Computer Science, P. Flocchini, J. Gao, E. Kranakis, and F. Meyer auf der Heide, Eds. Springer Berlin Heidelberg, 2014, pp. 51–66. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-45346-5_5

[11] S. McCourtney, *ENIAC, the triumphs and tragedies of the world's first computer*. United States of America: Walker Publishing, 1999.

[12] A. Adamatzky and J. Durand-Lose, "Collision-based computing," in *Handbook of Natural Computing*, G. Rozenberg, T. Bäck, and J. Kok, Eds. Springer Berlin Heidelberg, 2012, pp. 1949–1978. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-92910-9_58

[13] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, no. 3-4, pp. 219–253, 1982. [Online]. Available: http://dx.doi.org/10.1007/BF01857727

[14] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for your Mathematical Plays, 2nd edition*. A. K. Peters Ltd., 2001–2004.

[15] A. Adamatzky and P. Rendell, *Turing Universality of the Game of Life*. Springer London, 2002, pp. 513–539. [Online]. Available: http://dx.doi.org/10.1007/978-1-4471-0129-1_18

[16] R. A. Hearn, "The complexity of sliding-block puzzles and plank puzzles," *Tribute to a Mathemagician*, pp. 173–183, 2005.

[17] E. D. Demaine and R. A. Hearn, *Games of No Chance 3*. Mathematical Sciences Research Institute Publications, Cambridge University Press, 2009, vol. 56, ch. Playing Games with Algorithms: Algorithmic Combinatorial Game Theory, pp. 3–56. [Online]. Available: http://arXiv.org/abs/cs.CC/0106019

[18] E. Winfree, "Algorithmic self-assembly of DNA," Ph.D. dissertation, California Institute of Technology, June 1998.

[19] E. Winfree, F. Liu, L. Wenzler, and N. Seeman, "Design and self-assembly of two-dimensional DNA crystals," *Nature*, vol. 394, pp. 539–544, 1998.

[20] T. LaBean, E. Winfree, and J. Reif, "Experimental progress in computation by self-assembly of DNA tilings," *DNA Based Computers*, vol. 5, pp. 123–140, 1999.

[21] D. Doty, M. J. Patitz, and S. M. Summers, "Limitations of self-assembly at temperature 1," in *Proceedings of The Fifteenth International Meeting on DNA Computing and Molecular Programming (Fayetteville, Arkansas, USA, June 8-11, 2009)*, 2009, pp. 283–294.

[22] J. Maňuch, L. Stacho, and C. Stoll, "Two lower bounds for self-assemblies at temperature 1," *Journal of Computational Biology*, vol. 17, no. 6, pp. 841–852, 2010.

[23] P.-E. Meunier, M. J. Patitz, S. M. Summers, G. Theyssier, A. Winslow, and D. Woods, "Intrinsic universality in tile self-assembly requires cooperation," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 2014), (Portland, OR, USA, January 5-7, 2014)*, 2014, pp. 752–771.

[24] S. P. Fekete, J. Hendricks, M. J. Patitz, T. A. Rogers, and R. T. Schweller, "Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly," in *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015, to appear.